Algorithms Design and Analysis (CS500)

Instructor: Otfried Cheong

Class time: Wednesday, Friday 9:00–9:15

Course webpage: http://otfried.org/courses/cs500

Text book?

There is no text book—we will not follow a specific book. I will make slides available for each lecture and post links to on-line lecture notes where possible.

Piazza

You must regularly check the announcements on Piazza (see webpage). If you register there, they will be emailed to you automatically.

We will use Piazza for answering all your questions about the course contents. You can ask questions anonymously. You can ask questions in English or Korean.

TAs

박지원, 송현지, 도현우, 안성근.

Grading Policy

Participation (10%), Homework (20%), Midterm (30%), Final (40%).

Participation

Since the class is rather early, we will take attendance at the beginning of the class. You have three missed classes free—use this for doctor appointments, interviews, etc.

Homework

You can work in groups of up to three students, and each group only needs to submit one solution. Groups are allowed to change between homeworks.

Researching on the internet is discouraged. In any case, you or your group must write up your solution by yourself, and you must acknowledge all sources used for your solution (webpages, books, discussions with students).

Lecture schedule for the first two weeks:

- Wed, Mar 4 (today): First lecture
- Fri, Mar 6: Lecture by Eric Vigoda
- Wed, Mar 11: Lecture by Eric Vigoda
- Fri, Mar 13: No lecture
- Wed, Mar 20: Lectures continue normally.

Course Goals

- Appreciate the importance of algorithms in computer science and beyond (engineering, mathematics, natural sciences, social sciences, ...)
- Algorithmic thinking.
- Understand/appreciate limits of computation.
- Learn/remember some basic tricks, algorithms, problems.

What to expect

Comments from last year's evaluations:

- 과제가 너무 어려운 것 같습니다.
- 너무 어렵다. 너무 많은 내용을 진행하려고 한다.
- 이해하기 어려운 수업이었습니다. 학생들의 이해도에 맞게 진행하면 좋을 것 같습니다.
- 이해하기 어려운 수업입니다. 조금더 쉬운 방법으로 설명해주시면 좋을거 같습니다.
- Please explain in a lower pace especially for materials with a lot of variables/conceptually complicated.
- Class is too hard to understand, and the exam is too hard to solve.
- There were too many homeworks.
- i think this class needs a basic logic class as a prerequisite class. most of the student who studied in different university doesn't have enough background to follow this class.

What to expect

But other comments:

- I was hoping we can cover more topics in this course , also the lecture should be a little bit faster since it's advance course in graduate level.
- I'd be happier if you skip easy things and speed up a little bit so that more things can be covered.

Fact: CS500 is a rather easy "Graduate Algorithms" course. Have a look at the webpage of graduate algorithms at UIUC or CMU—they cover twice as much material in more depth.

Students who want to learn deeply about algorithms will be disappointed. Sorry. You can better take CS422 (Theory of Computation) or CS492 (Advanced Algorithms).

What to expect

I suggested that the Department removes the theory requirement.

We cover only topics that are important and useful to all graduate students in CS, if they are interested in theory or not.

We go slowly and explain in detail.

Nevertheless, this is a graduate course:

- Don't complain that there is no text book that we follow closely. As a graduate students, you are expected to read original research papers.
- We must assume some prior knowledge about "basic logic" and algorithms. If you had no undergraduate algorithms course, audit CS300 before taking CS500.

Requirements

- Big-Oh notation, basics of asymptotic analysis
  - Definition of O, Ω, Θ
  - Can you analyze merge sort?
  - To refresh, read Chapter 1 of Jeff Erickson's lecture notes.
- Graphs
  - Definition of graphs, paths, trees, connected components, etc.
  - Breadth-first search and depth-first search
  - Directed graphs and topological ordering
  - To refresh, read Chapters 18 and 19 of J.E.'s lecture notes.
- Divide-and-conquer, Dynamic Programming

Are you ready for this course?

Homework #0 has been posted on the course website.

It allows you to see if you have the mathematical maturity to take this course. Please have a look at the problems today. If the problems look very hard/impossible to you, you should probably audit CS300 this semester and take a graduate theory course later.

Take this course seriously

Some students think: This course is too difficult for me, I will not even try. Nobody ever fails in graduate courses and I cannot get an A anyway, so I'm just okay with a B-.

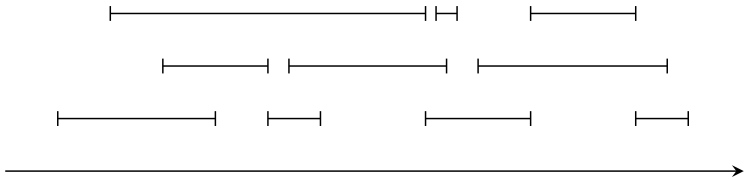
Students who make no effort and receive less than, say, 5 of 100 points for midterm and final, will fail.

Interval Scheduling

Input: A set of jobs with start and finish times to be scheduled on a resource

Goal: Schedule as many jobs as possible

Two jobs with overlapping intervals cannot both be scheduled!

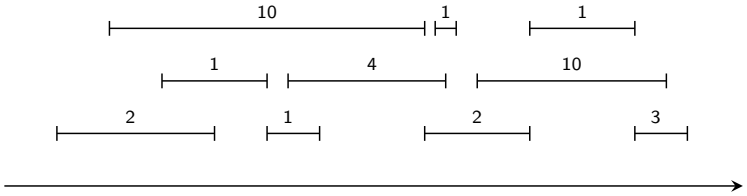


Weighted Interval Scheduling

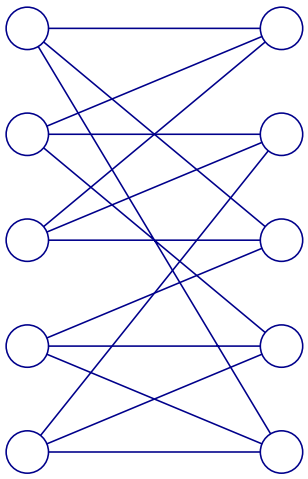
Input A set of jobs with start times, finish times and weights

Goal Schedule jobs so that total weight of jobs is maximized

- Two jobs with overlapping intervals cannot both be scheduled!



Bipartite matching



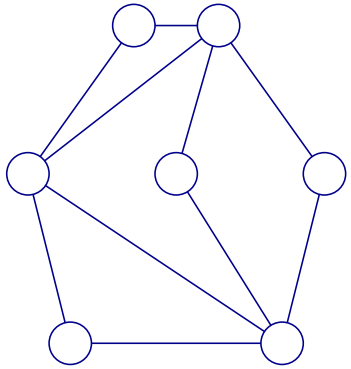
Input: A bipartite graph
Goal: A matching of maximum cardinality

Matching: Subset of edges such that every vertex has at most one edge incident upon it.

Independent Set

Input: A graph
Goal: An independent set of maximum cardinality.

Independent Set: Subset of vertices such that no two are joined by an edge.

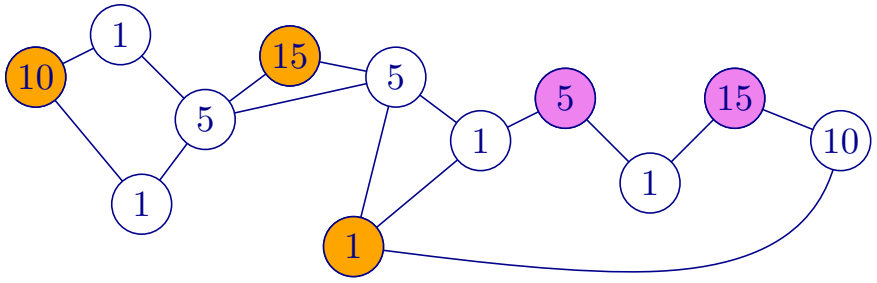


Competitive Facility Location

Input: Graph with weight on each vertex
Game: Two players alternately pick vertices such that

1. The set of picked vertices is an independent set
2. Players want to maximize the total weight of vertices they picked

Question: Does player one have a winning strategy?



Five representative problems

Interval scheduling: greedy algorithm

Weighted interval scheduling: dynamic programming

Bipartite matching: polynomial time algorithm, for instance using max-flow

Independent Set: NP-complete problem, no subexponential algorithm known

Competitive Facility Location: PSPACE-complete

These problems are related. Relationships between problems are formally discussed using reductions.

Problem Types

The world is full of algorithmic problems:

- decision problems (example: does player one have a winning strategy)
- search problems (example: find a winning strategy)
- optimization problems (example: what is the size of the largest independent set in a graph)

When comparing the difficulty of problems, it is easiest to think about decision problems.

For each search and optimization problem, we can define a corresponding decision problem. It is not harder than the original problem. Example: Given a graph G and an integer k, does G have an independent set of size at least k.

Problems and Instances

An instance of BipartiteMatching is a bipartite graph G, and an integer k. The solution to this instance is "YES" if G has a matching of size $\geq k$, and "NO" otherwise.

A problem is a set of instances.

An algorithm for a decision problem X takes as input an instance of X, and returns "YES" or "NO" as output.

Efficient Algorithms

In this class efficiency is broadly equated to polynomial time:

$O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, $O(n^{100})$,...

where n is the size of the input.

Why? Is n^{100} really efficient/practical? Etc.

Short answer: polynomial time is the single, robust, mathematically sound way to define efficiency.

Approximation algorithms

When we cannot solve an optimization problem efficiently, we can still try to come up with an approximation algorithm.

Let's call the value of the optimal solution to an instance X of an optimization problem $Opt(X)$.

An algorithm A for a minimization problem is an α-approximation with approximation factor $\alpha \geq 1$ if

$$A(X) \leq \alpha \cdot Opt(X)$$

An algorithm A for a maximization problem is an α-approximation with approximation factor $\alpha \leq 1$ if

$$A(X) \geq \alpha \cdot Opt(X)$$

Vertex Cover

A vertex cover for a graph $G = (V, E)$ is a subset $C \subseteq V$ such that every edge in E has at least one endpoint in C.

The MinVertexCover problem asks for the smallest vertex cover for a given graph G.

A simple greedy heuristic: Pick a vertex with highest degree, remove all incident edges, and repeat until no edges are left.

Does this compute the minimal vertex cover?

Is it an α-approximation algorithm, for some constant $\alpha > 1$?

A simple 2-approximation

ApproximateVertexCover(G):
  $S \leftarrow \emptyset$
  while $E(G) \neq \emptyset$:
    Pick any edge $uv \in E(G)$
    $S \leftarrow S \cup \{u, v\}$
    Remove alle edges incident to u and v from G.
  return S

Theorem: This algorithm is a 2-approximation algorithm for VertexCover.

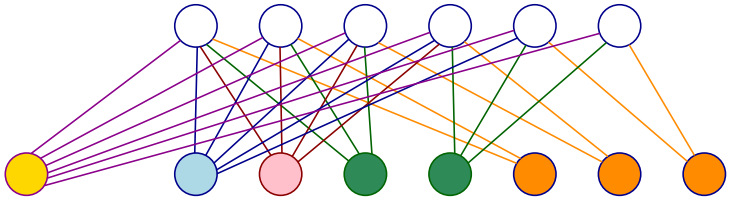
Proof: The edges uv selected by the algorithm form a matching in the original graph G.

Greed is Bad

Here is an instance where the greedy algorithm does badly: A bipartite graph $(U \cup V, E)$.

U consists of n nodes.

V consists of $n - 1$ groups: For i from 2 to n, make $\lfloor n/i \rfloor$ nodes connected to i nodes of U.



Optimal vertex cover is U (of size n), but greedy algorithm chooses V (of size $\Theta(n \log n)$).